T. Born 2 - 1

Title: **Automatic Compilation of Electronic Telecommunications System Message Generation Code**

Technical Field:

This invention relates to the automatic generation of message generation code that is in accordance with the protocol specified by the ASN.1 (Abstract Syntax Notation.1), requirements of the International Telecommunications Union (ITU), and the International Standards Organization (ISO).

Problem:

The messages generated by electronic telecommunications systems are specified in an Abstract Syntax Notation (called ASN.1 - a definition for which is specified in ITU documents X.680, 1, 2, & 3). Manufacturers throughout the world are required to describe and negotiate signaling between such electronic devices using the ASN.1 syntax; otherwise, many carriers would refuse to purchase machines from vendors who cannot specify the communications between their machinery and associated interacting devices in accordance with these standards.

In the prior art, there is no fully automatic system available for converting the description of the communication into code that executes that communication, and is made available for Programmers to incorporate that code into a (Computer Aided Software Engineering) CASE design tool, in order that the communication aspect of a system can be incorporated into a larger system design. In one prior art system, the ASN.1 specification is compiled in an ASN.1 Compiler, whose output is then fed into an Object Machine Compiler; the output of this stage is modified by an Application Programmer (in person) to be fed into the development environment, such as provided by the aforementioned CASE Tool. One such tool is ObjecTime, provided by the Rational Corporation. However, when this CASE Tool is used, a large body of Auxiliary Statements are required that have to be written by the Applications Programmer, to incorporate the output of the Object Machine Compiler into the design. The need for this extensive manual input lowers the integrity of the final product, because of the possibility of introducing errors in writing these Auxiliary Statements. It also requires a large amount of effort to generate these Auxiliary Statements manually. Further, if the manufacturer, or the telecommunications community, modifies the specification of the communication between those telecommunication devices, then a large amount of additional effort would be required.

Summary of the Invention:

The above problem is solved and an advance is made over the prior art in accordance with this invention, wherein the CASE Tool receives the output of a new ASN.1 Tool, called ASYN, and removes the need for manual intervention of the Applications Programmer, thus allowing the CASE Tool to generate all the output required that can be compiled by an Object Machine Compiler. Advantageously, the use of ASYN permits the by-passing of manual effort in processing the ASN.1 statements to produce Auxiliary Statements for use by the CASE Tool, and allows a new ASN.1 description to be automatically compiled. ASYN does not rely upon the specialized knowledge of the Applications Programmer, or the reformatting of the ASN.1 into a tabular form to aid the comprehension of the Applications Programmer, in representing the ASN.1 formatted standard.

The external input to ASYN is the ASN.1 Source File. It contains the ASN.1 statements that define the communication protocol between telecommunications systems. An ASN.1 Lexical Analysis breaks down the ASN.1 source file into individual lexemes, words and other tokens. The ANS.1 Lexemes are processed by an ASN.1 Syntax Analysis process into an ASN.1 Data Repository. This analysis process guarantees that the ASN.1 input file is syntactically correct and deduces the meaning or vital information about each source line. This information is stored in a logical order in the ASN.1 Data Repository. This Data Repository is a Data Scheme that contains the meaning and associations of the ASN.1 Source File without any of its syntactic complexity.

Another input to the ASYN process is a GenScript Source configuration file. This source file contains the rules for creating any information required from data held in the ASN.1 Data Repository. This information can be created in any format. If the CASE Tool is changed, and an alternative output is required, then it is the GenScript that will require modification. The RULES found in GenScripts are a set of procedural statements that describe how to convert from the raw data in the ASN.1 Data Repository into contextualized information in a CASE Tool Intermediate file. A GenScript Lexical Analysis procedure splits the GenScript file into lexemes, words, and tokens, and stores them in an ordered list of GenScript Lexemes. The GenScript Lexemes are subject to the GenScript Syntax and semantic analysis. This process guarantees that the GenScript input is syntactically correct and converts its meaning into an interpretable internal format, GenScript Pre-compiled Functions. This format is interpreted by a virtual machine (GenScript Executor). The virtual machine executes the set of pre-compiled statements that allow the execution of sequences (lists of consecutive statements), conditional sequences, (a set of statements executed upon a condition being met), and looping, (the repetition of a set of statements while a condition is valid), that is contained in the GenScript Source. The GenScript Executor executes the compiled statements and operates upon the data found in the ASN.1 Data Repository to create the CASE Tool Intermediate File.

Applicants have inventively combined the analysis of ASN.1 source with pre-compiled functions generated from a Genscript source as inputs to a Genscript Executor; the output of this Genscript Executor then compiled by an Intermediate CASE tool to act as the input to a target machine compiler.

A data processing system is used to run the programs discussed above to produce the desired machine code.

Brief Description of the Drawing(s):

Figure 1 is a block diagram illustrating the prior art;

Figure 2 is a block diagram illustrating the operation of the Applicants' invention; and

Figure 3 is a block diagram illustrating the details of Applicants' ASN.1 Tool, ASYN.

Detailed Description:

Figure 1 is a block diagram illustrating the overall process of the prior art. It consists of an ASN.1 Compiler (process 1.4), a development environment (or CASE Tool (process 1.5), and an Object Compiler (process 1.9). The inputs to the development environment are the Application Programmer's input describing what is to be performed on acceptance of messages contained in the protocol (definition 1.3) to produce object code (file 1.8) - usually generated in the C or C++ languages. The ASN.1 Compiler receives as input the full complement of ASN.1 statements to define the communication protocol into and out of a telecommunications system (file 1.1), so that it might be designated (by the industry) as a standard. The ASN.1 statements are compiled using an existing compiler to produce object code (file 1.6) - usually generated in C or C++. The Applications Programmer also needs to create/ generate/invent (process 1.2), extra "glue" object code (file 1.7) to "tie together" (encode and decode instances of messages defined in the ASN.1 definition), the code generated by processes 1.4 and 1.5 in files 1.6 and 1.8. This is then all compiled together (process 1.9) to produce the final executable code (file 1.10).

Figure 2 is a diagram showing the operation of Applicants' invention, whose object is to avoid the repetitive and error prone manual effort required to generate C or C++ Auxiliary Statements to encode and decode instances of messages defined in the ASN.1 definition. In the new system, the CASE Tool (process 2.4) receives two inputs: the Application Programmer's's input (definition 1.3), previously described with respect to Figure 1, and the output of an ASYN Tool (process 2.2). The inputs to the ASYN Tool (file 1.1) are identical to the input to the ASN.1 Compiler (process 1.4, Figure 1). The output from the ASYN Tool (file 2.5) is an intermediate file that is an input to the CASE Tool, and is generated in a language that is specific to the CASE Tool. The CASE Tool now generates the C or C++ object code (file 2.6), previously generated by the ASN.1 Compiler (process 1.4, Figure 1), the Application Programmer (process 1.2, Figure 1), and the CASE Tool itself, (process 1.5, Figure 1). The Objet Machine Compiler (process 1.9) then uses the object code generated by the CASE Tool as in the prior art.

Figure 3 is a diagram illustrating the operation of the ASYN Compiler. The external input to ASYN is (file 1.1), the ASN.1 Source File (file 1.1, Figure 1, and file 1.1, Figure 2). It contains the ASN.1 statements that define the communi-cation protocol between telecommunications systems. The ASN.1 Lexical Analysis (process 3.3) breaks down the ASN.1 source file into individual lexemes, words and other tokens. Lexemes are consecutive groups of characters from a source file that are logically grouped together to produce a "token". Tokens can be numeric (0x13b2, 0167, 12394), alphanumeric (name, file 1, size 3), symbolic (+, *, +=), or separators

<TAB>, <SPACE>, <CRLF>). These are stored in an ordered list of ASN.1 Lexemes (repository 3.4). The ANS.1 Lexemes are processed in the ASN.1 Syntax Analysis (process 3.5) into the ASN.1 Data Repository (repository 3.6). This process guarantees that the ASN.1 input file is syntactically correct and deduces the meaning or vital information about each source line. This information is stored in a logical order in the ASN.1 Data Repository (3.6). This Data Repository is a Data Schema that contains the meaning and associations of the ASN.1 Source File without any of its syntactic complexity.

An input to the ASYN process is the GenScript Source configuration file (file 3.7). This source file contains the rules for creating any information required from data held in the ASN.1 Data Repository. This information can be created in any format. If the CASE Tool is changed, and an alternative output is required, then it is the GenScript that will require modification. The RULES found in GenScripts are a set of procedural statements that describe how to convert from the raw data in the ASN.1 Data Repository (repository 3.6), into contextualized information in the CASE Tool Intermediate (file 2.5), e.g., "for each data type in the project, produce an ObjecTime data definition". The GenScript Lexical Analysis procedure (process 3.8) splits the GenScript file into lexemes, words, and tokens, and stores them in an ordered list of GenScript Lexemes (repository 3.9). The GenScript Lexemes are subject to the GenScript Syntax and semantic analysis (process 3.10). This process guarantees that the GenScript input is syntactically correct and converts its meaning into an interpretable internal format (GenScript Pre-compiled Functions (repository 3.11)). This format is interpreted by a virtual machine (GenScript Executor (process 3.12)). The virtual machine executes the set of pre-compiled statements that allow the execution of sequences (lists of consecutive statements), conditional sequences, (a set of statements executed upon a condition being met), and looping, (the repetition of a set of statements while a condition is valid), that is contained in the GenScript Source (file 3.7). The GenScript Executor executes the compiled statements and operates upon the data found in the ASN.1 Data Repository (repository 3.6), to create the CASE Tool Intermediate File (file 2.5).

The programs discussed above are executed on a data processing system to compile the desired executable code for a target switching machine.

The above description is of one preferred embodiment of Applicants' invention. Many other embodiments will be apparent to those of ordinary skill in the art without departing from the scope of the invention. The invention is limited only by the attached Claims.